March 1987

UILU-ENG-87-2218
CSG-66

COORDINATED SCIENCE LABORATORY
College of Engineering

PERFORMANCE ANALYSIS OF NUMERICAL PROBLEMS ON A LOOSELY COUPLED SYSTEM

Joseph T. Rahmeh
Jacob A. Abraham

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-87-2218 (CSG-66)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Semiconductor Research Corporation (SRC)* Texas Instruments, Inc. (TI) **
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) Research Triangle Park * Dallas ** North Carolina Texas 27709 75265
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SRC TI 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER 86-12-109 (SRC) none (TI)
8c. ADDRESS (City, State, and ZIP Code) Research Triangle Park Dallas North Carolina Texas 27709 75265	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Performance Analysis of Numerical Problems on a Loosely Coupled System

12. PERSONAL AUTHOR(S)
Rahmeh, Joseph T. and Abraham, Jacob A.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	March, 1987	28

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	distributed systems; load balancing; modeling; simulation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

We present a performance model for a class of numerical problems on a loosely coupled system of processors. We validate the model by comparing its performance predictions to empirical data collected from the implementation of matrix multiplication, FFT, and two-dimensional FFT on a network of workstations. We show that the large communication overhead limits the speedup and, for a fixed-size problem, may result in a decrease of speedup as the number of processors assigned to the problem is increased. We use the model to study the effects of broadcasting and the use of a communication processor on the performance of the system.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

Performance Analysis of Numerical Problems [†] on a Loosely Coupled System

Joseph T. Rahmeh
and
Jacob A. Abraham

Computer Systems Group
University of Illinois
Coordinated Science Laboratory
1101 W. Springfield
Urbana IL 61801

[†] This work was supported by the Semiconductor Research Corporation under contract 86-12-109 and by a grant from Texas Instruments, Incorporated.

1. Introduction

Distributed computer systems consisting of a collection of workstations and mainframe machines connected through a local area network are becoming a standard way of providing computer service. Such systems provide resource sharing, good performance, highly interactive user interfaces, and possibly improved reliability over centralized systems. Stankovic [1] gives an excellent survey of distributed computer systems.

Good performance in distributed computer systems results from resource dedication and/or load balancing. Resource dedication calls for reserving a resource or set of resources for the exclusive use of one task (or job) in order to minimize its turnaround time. Load balancing, on the other hand, consists of moving tasks or subtasks from a loaded resource to unused or slightly loaded resource in order to maximize the throughput of the whole system. Note that in the case of a single job and an idle system, resource dedication and load balancing are identical.

In this report we consider the performance of a single job distributed on a cluster of workstations dedicated to that job. Our work is motivated by the abundance of computation-intensive jobs and the availability of workstations. Dedicating otherwise idle (at night time for example) workstations to a computation intensive job may offer a low cost solution to expensive problems.

Performance of distributed systems may be obtained through one of three methods: modeling, simulation, and direct measurement [2]. These methods provide a range of flexibility and accuracy. In general, modeling is the least accurate since it requires many simplifying assumptions in order to make the model tractable. Direct measurement offers the most accuracy, however the results are limited to the system being measured.

In this report we couple modeling and measurements and obtain an accurate model by:

(1) concentrating on specific algorithms (this results in a simple model),

(2) embedding measured system parameters into the model, and

(3) validating the model by comparing the performance it predicts to experimental data.

We then use the validated model to study the effects of communication [3-5] on the performance of the system and to determine the conditions under which the communication overhead reduces the speedup as more processors are directed to the problem.

The remainder of the report is organized as follows. In section two we derive the performance model of the system. In section three we validate the model by comparing model-predicted and experimental results. In section four we use the validated model to study the effects of varying system parameters on performance. Section five contains the conclusions.

2. Performance Model

The system we are modeling consists of a set of identical workstations connected through an single bus (ethernet cable). Let P denote the number of processors participating in the experiment, and T and T_P denote the task turnaround time on one and P processors, respectively. The performance measures we are interested in are the speedup,

$$\sigma=\frac{T}{T_P},$$

and the efficiency,

$$\epsilon=\frac{\sigma}{P}.$$

The speedup of a task (or job) depends on the task, how the task is partitioned into subtasks, and how the subtasks are allocated to the participating processors. We make the following assumptions about the task and about the partitioning and allocation issues:

(1) The job originates at one processor hereafter referred to as the primary processor. Any data pertaining to the job is originally available only on the primary processor.

(2) The job finishes on the primary processor. All results reside on the primary processor when the job terminates.

(3) All other processors, called secondary processors, on the network are either idle or cooperating with the primary processor on the distributed job.

(4) The algorithm is perfectly orchestrated: there is no contention for the network.

(5) The job consists of two or more phases, each running either serially on the primary processor or in parallel on all of the participating processors. We consider jobs with only one parallel phase since the general case may be regarded as a sequence of subjobs, each containing exactly

one parallel phase.

(6) In a parallel phase the primary processor splits the data between itself and the secondary processors, sends portions of the data to the secondary processors, computes its portion of the problem, polls the secondaries for subresults, and merges the subresults.

We further simplify the analysis by concentrating only on the parallel phase of the job. The speedup and efficiency of the whole job can be easily determined from those of the parallel phase.

2.1. Model Parameters and Equations

First, we define the model parameters for the parallel phase of the algorithm. Let

(1) n denote the size of the the problem,

(2) P the number of processors,

(3) t the turnaround time on a single processor (T denotes the single processor turnaround time for the whole job),

(4) t_P the turnaround time on P processors (T_P is used for the whole job),

(5) τ_i the execution cost function on processor i,

(6) S_i the size of data sent to processor i,

(7) R_i the size of data received from processor i,

(8) C the communication cost function (e.g., $C(m)$ is the cost of sending m bytes from any processor to any other processor assuming no contention for the ethernet),

(9) W_i the wait time for results from secondary processor i (e.g W_i is the time the primary processor blocks waiting for secondary processor i to finish its computation) and, finally,

(10) M the cost to merge the subresults.

Note that the wait parameter is redundant since it can be obtained from the execution time and

communication (send and receive) parameters.

The network turnaround time is the sum of the time to send the data to all the secondary processors, the computation time on the primary processor, the time to wait for and receive subresults from the secondary processors, and the time to merge the subresults:

$$t_P = \sum_{i=1}^{P-1} C(S_i) + \tau_0 + \sum_{i=1}^{P-1} (W_i + C(R_i)) + M. \tag{2.1}$$

Note that the primary processor is processor 0. Any analysis of t_P requires the knowledge of the partitioning (S and R terms) of the problem and the execution cost (τ functions) of each of the partitions. We first consider the case where the secondary processors each receive the same amount of data and where the execution cost function does not depend on the data or the processor.

2.2. Even Partitioning Model

In the even partitioning model we assume that the secondary processors receive the same amount of data and send back the same amount of results. We also assume that the execution cost functions are identical and independent of the processor and of the processed data. The partitioning strategey is what Dubois and Briggs [6] refer to by "vectorial decomposition". Figure 2.1 shows a timing diagram fitting the above assumptions for a system consisting of four processors. Note that the primary processor begins and finishes execution at the same time as the last secondary proces-

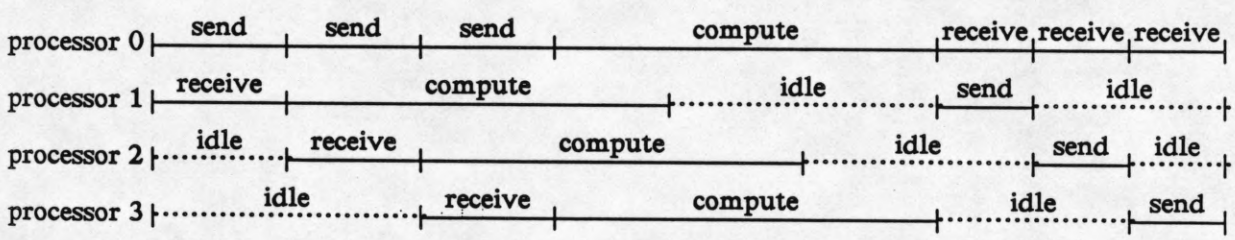

Fig. 2.1. Timing diagram for even partitioning.

sor. Therefore, all secondary processors finish at the same time as or before the primary processor and the wait terms vanish from the network time equation. The network time equation becomes:

$$t_P = (P-1)C(S) + \tau_0 + (P-1)C(R) + M$$
$$= \tau_0 + (P-1)\left[C(S) + C(R)\right] + M \qquad (2.2)$$

which results in the following speedup (we use σ_0 for the parallel phase and σ for the whole job):

$$\sigma_0 = \frac{t}{t_P} = \left[\frac{\tau_0}{t} + \left\lceil\frac{(P-1)(C(R)+C(S)) + M}{t}\right\rceil\right]^{-1}.$$

Note that, ideally, we would like the speedup to be t/τ_0. The extra terms in the speedup equation result from the merge time and the idle segments in Figure 2.1. The idle segments in Figure 2.1 result from two factors:

(1) The primary processor communicates with the secondary processors one processor at a time. Thus, when data is sent to the first secondary processor, the rest are idle. We will consider ways to avoid such a serialization through broadcasting in section 4.

(2) The primary processor starts its computation last; thus, the secondary processors finish first and have to be idle while waiting for the primary processor to receive their results. In section 4 we will show how to avoid idling the secondary processors through the use of a communication processor.

2.3. Relating the Parallel Phase to the Whole Job

We now establish the relationship between the speedup of the parallel phase and that of the whole job. Recall that T and T_P denote the turnaround time of the whole job on one and P processors, respectively. Let γ denote the fraction of T which is not parallelizable, i.e., γT represents the execution cost of the portion of the job that is not distributed. We have:

$T = \gamma T + t$ and $T_P = \gamma T + t_P$;

therefore,

$$\sigma = \frac{T}{T_p} = \frac{T}{\gamma T + t_P} = \frac{T}{\gamma T + \frac{t}{\sigma_0}} = \frac{T}{\gamma T + \frac{(1-\gamma)T}{\sigma_0}}$$

$$= \frac{1}{\gamma + \frac{(1-\gamma)}{\sigma_0}}. \qquad (2.3)$$

For a fixed job size and varying number of processors assigned to that job, we now establish that, if the speedup of a job peaks as we increase the number of processors assigned to it, then so does the speedup of the parallel phase and at exactly the same number of processors. The proof hinges on the assumption that, for a fixed job size, the serial fraction γ is independent of the number of processors P:

LEMMA 2.1: The speedup of a job σ and the speedup of its parallel phase σ_0 assume their maximum (local or global) at the same number of processors P.

PROOF: If $\gamma = 0$ then $\sigma = \sigma_0$ and the lemma is trivially valid.

If $\gamma = 1$ then $\sigma = 1$ and $\sigma_0 = 0$ and the lemma is, again, trivially valid.

Otherwise, $0<\gamma<1$. The derivative of σ with respect to P is:

$$\frac{d\sigma}{dP} = \frac{\frac{-d}{dP}\left[\frac{(1-\gamma)}{\sigma_0}\right]}{\left[\gamma + \frac{(1-\gamma)}{\sigma_0}\right]^2} = \frac{(1-\gamma)}{{\sigma_0}^2\left[\gamma + \frac{(1-\gamma)}{\sigma_0}\right]^2}\frac{d\sigma_0}{dP} \qquad (2.4)$$

which implies that, for each value of P, the derivatives of σ and σ_0 either have the same sign or are both zero.

3. Implementation

3.1. Introduction

In this section we describe the distributed, even partitioning implementation of matrix multiplication, Fast Fourier Transform (FFT), and 2-dimensional Fast Fourier Transform (2d FFT). We apply the model derived in section two to each of the above problems and compare the performance predicted by the model to the one measured from the implementations. Experimental data was collected on a cluster of five SUN-Microsystems workstations (model 3/50) all equipped with a floating point accelerator (FPA) chips and running the UNIX operating system. All experiments were repeated with the FPA turned off to emulate a system with a better communication to computation ratio. Distribution was achieved through the use of the remote procedure call, RPC, programming paradigm. First, we describe the experiments used to determine the key parameters of the model.

3.2. Measuring Model Parameters

The parameters of the model consist of the size of the data sent to and received from the secondary processors, the computation cost function, the communication cost function, and the cost of merging the subresults received from the secondary processors. Since we are considering even partitioning, the size parameters are independent of the secondary processors, and the computation cost function is independent of the processors and the data.

The size parameters are obtained directly from the algorithm, as will be seen later. The computation cost function is obtained from the complexity analysis of the algorithms, and the coefficients of that function are evaluated by fitting the function to measured data. For example,

the complexity analysis of the FFT yields a cost function of $\alpha n \log n$†. We evaluate α by measuring the cost of the FFT (versus n) and then fitting the measured data to the function $\alpha n \log n$.

Since the communication medium is a packet-oriented bus [7], the communication cost function is hypothesized to have the form:

$$C(m) = a_0 \left\lceil \frac{m}{s} \right\rceil + b_0 m + l \tag{3.1}$$

and is approximated with:

$$C(m) \approx (\frac{a_0}{s} + b_0)m + l = am + l \tag{3.2}$$

where l is the communication latency due to the operating system call and network delay overheads, s is the packet size, and a_0 and b_0 are the communication cost per packet and per byte respectively. The packet size was obtained from an operating system constant table. The coefficients a_0, b_0, and l were obtained empirically through simple network experiments. Replacing the coefficients of equation 3.2 by their values, we get ($C(m)$ is expressed in milliseconds):

$$C(m) \approx 7.93e{-}3m + 9.62 \tag{3.3}$$

Figure 3.1 validates equation 3.3 by showing how well it fits the measured data. Figure 3.2 shows the transmission rate (both measured and computed) versus the message size. Note that the best measured transmission rate of 122 Kbytes/s falls much lower than the rate of 1.25 Mbytes/s which a 10 Mbits/second ethernet is capable of delivering.

3.3. Model Validation

In this section we consider distributed implementations of matrix multiplication, Fast Fourier Transform, and 2-dimensional Fast Fourier Transform problems. For each job, we derive a speedup equation for the parallel phase and then use that to determine the speedup of the whole job (t and σ_0 express the the time and speedup of the parallel phase and T and σ those of the whole

†All logarithms are to base two.

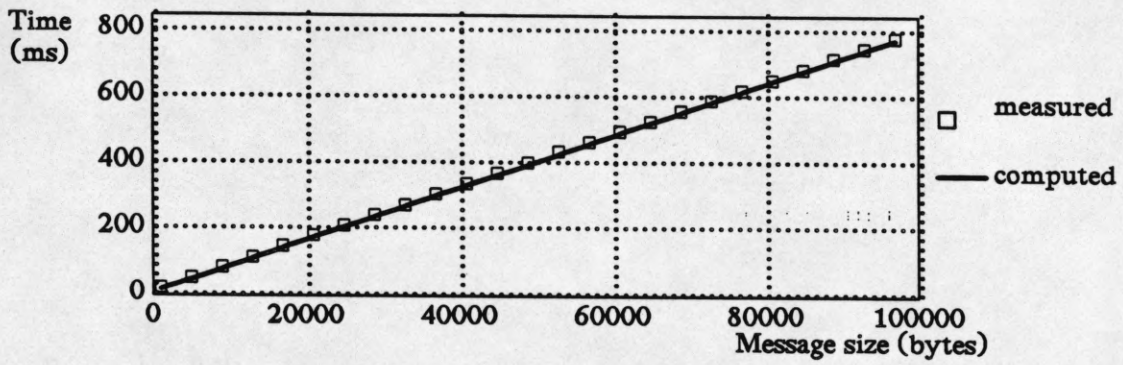

Fig. 3.1. Fit of communication function.

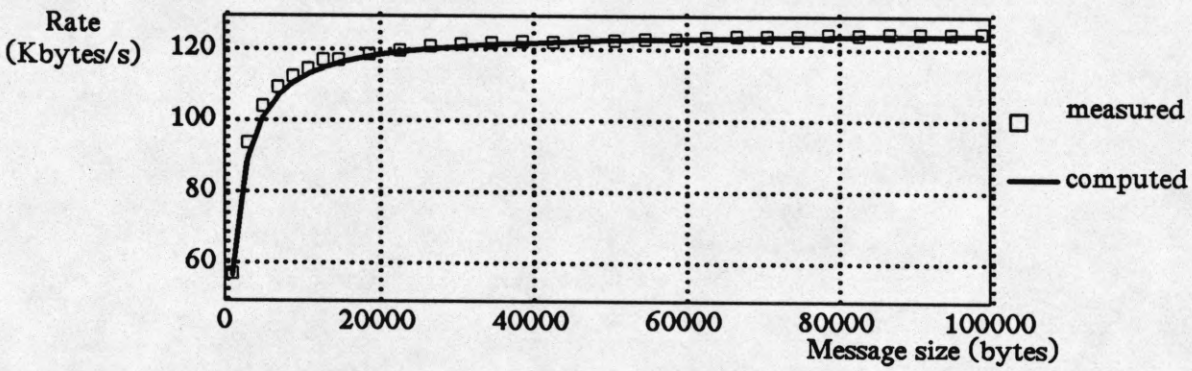

Fig. 3.2. Communication rate versus message size.

job).

3.3.1. Matrix Multiplication

The objective is the product of two matrices:

$$\underline{A} = \underline{B} \times \underline{C}$$

where $\underline{A}$, $\underline{B}$, and $\underline{C}$ are $u \times u$ matrices (extension to non-square matrices is trivial). Each of the processors participating in the distributed algorithm computes u/P rows of $\underline{A}$ (assume for a moment that u is a multiple of P). The distributed algorithm consists of three steps:

(1) The primary processor ships $\underline{C}$ and u/P rows of $\underline{B}$ to each secondary processor; a secondary processor starts computing as soon as it receives data.

(2) The primary processor computes its u/P rows of $\underline{A}$.

(3) The primary processor receives the results of the secondary processors into their appropriate positions in the $\underline{A}$ matrix. Thus, the cost of merging the subresults is zero.

The size of the problem is $n=u^2$. The single processor computation cost function is:

$$t = \alpha n^{1.5} \tag{3.4}$$

where α is empirically determined to be 1.302e-5 (1.6e-4 without the FPA). The primary processor sends $\underline{C}$ and u/P rows of $\underline{B}$ to and receives u/P rows of $\underline{A}$ from each secondary processor, thus:

$$S = n+\frac{n}{P} \text{ and } R = \frac{n}{P}. \tag{3.5}$$

It is understood that S and R are expressed in bytes, even though we do not include any conversion (to bytes) factors in their expressions.

The computation cost function in the distributed algorithms is (cost of evaluating u/P rows of $\underline{A}$):

$$\tau = \frac{\alpha n^{1.5}}{P}. \tag{3.6}$$

The network time function can now be written as:

$$t_P = \frac{\alpha n^{1.5}}{P} + (P-1)\left[C\left[n+\frac{n}{P}\right] + C\left[\frac{n}{P}\right]\right] \tag{3.7}$$

which correspond to a speedup of:

$$\sigma_0 = \left\{\frac{1}{P} + \frac{(P-1)}{\alpha n^{1.5}}\left[C\left[n+\frac{n}{P}\right] + C\left[\frac{n}{P}\right]\right]\right\}^{-1} \tag{3.8}$$

which is identical to the speedup, σ of the whole problem since matrix multiplication contains no serial phase. Figure 3.3 shows both the measured and computed speedup versus the number of rows in the matrix.

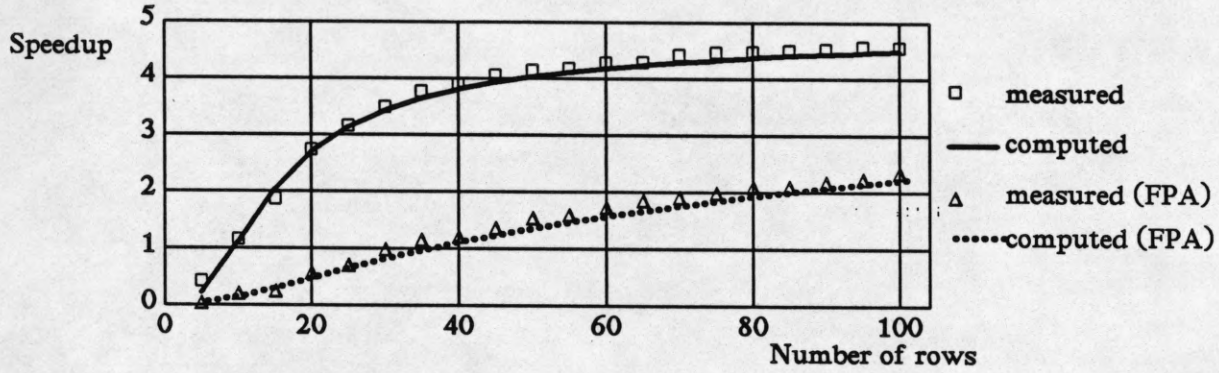

Fig. 3.3. Speedup versus number of rows for matrix multiplication.

3.3.2. Fast Fourier Transform

The Discrete Fourier transform (DFT) [8] maps the vector $\vec{X} = (x_0, x_1, \cdots x_{n-1})$ into the vector $\vec{Y} = (y_0, y_1, \cdots y_{n-1})$ such that

$$y_k = \sum_{m=0}^{n-1} x_i e^{-2jmk\pi/n} \tag{3.9}$$

A straightforward implementation of the DFT requires $O(n^2)$ steps. However, the Fast Fourier Transform algorithm (FFT) [8-9] computes the DFT in $O(n \log n)$ steps. The FFT consists of $\log n$ stages which transform $\vec{X}$ into $\vec{Y}$. Each stage consists of $n/2$ butterfly operations on the output of the previous stage (the first stage operates on $\vec{X}$). We modify the FFT algorithm such that, at each stage, the two operands of each butterfly operation are adjacent. Figure 3.4 contains a description of the modified algorithm. The "bit_reversal_permutation" and the "shuffle" functions guarantee that the operands of each butterfly operation are adjacent.

$\vec{A}$ = bit_reversal_permutation($\vec{X}$)
for i=1 to log n do
    $\vec{B}$ = parallel_butterfly($\vec{A}$)
    $\vec{A}$ = shuffle($\vec{B}$)
end for
$\vec{Y}$ = shuffle($\vec{B}$)

Fig. 3.4. Parallel FFT algorithm.

The parallel implementation of the modified FFT algorithm distributes the $n/2$ butterfly operations among the participating processors. The function "parallel_butterfly" is the only distributed portion of the modified algorithm; its single processor computation cost function (evaluation of $n/2$ butterfly operations) is:

$$t = \alpha n \tag{3.10}$$

During one execution of "parallel_butterfly", each secondary processor receives $1/P$ of $\vec{A}$ and $1/(2P)$ of a vector of coefficients needed to perform the butterfly operations and returns $1/P$ of $\vec{B}$ thus:

$$S = 1.5\frac{n}{P} \text{ and } R = \frac{n}{P} \tag{3.11}$$

The computation cost function on each of the secondary processors is (cost of evaluating $n/(2P)$ butterfly operations):

$$\tau = \frac{\alpha n}{P}. \tag{3.12}$$

The network time function becomes:

$$t_P = \frac{\alpha n}{P} + (P-1)\left[C(1.5\frac{n}{P}) + C(\frac{n}{P})\right] \tag{3.13}$$

which yields a speedup of:

$$\sigma_0 = \left[\frac{1}{P} + \frac{P-1}{\alpha n}(C(1.5\frac{n}{P}) + C(\frac{n}{P}))\right]^{-1}. \tag{3.14}$$

To obtain the FFT speedup, σ, from the parallel phase speedup, σ_0, all we need is to compute the fraction of non-parallelized code γ. The serial FFT time is $\beta n \log n$, and the vector butterfly operation is performed $\log n$ times thus:

$$\gamma = \frac{\beta n \log n - \alpha n \log n}{\beta n \log n} = \frac{\beta - \alpha}{\alpha} \tag{3.15}$$

Beta and alpha are determined empirically. Figure 3.5 shows the measured and computed speedups of the distributed FFT algorithm for a range of number of points.

3.3.3. Two Dimensional FFT

The two-dimensional discrete Fourier transform [8] maps the matrix $\underline{X} = [x_{lm}]_{u \times u}$ into the matrix $\underline{Y} = [y_{lm}]_{u \times u}$ such that

$$y_{rv} = \sum_{l=0}^{u-1} \left(e^{-2jrl\pi/u} \sum_{m=0}^{u-1} x_{lm} e^{-2jvm\pi/n} \right). \tag{3.16}$$

One implementation [8] of the 2d-DFT consists of the following three steps:

(1) Perform the FFT on each of the rows of $\underline{X}$.

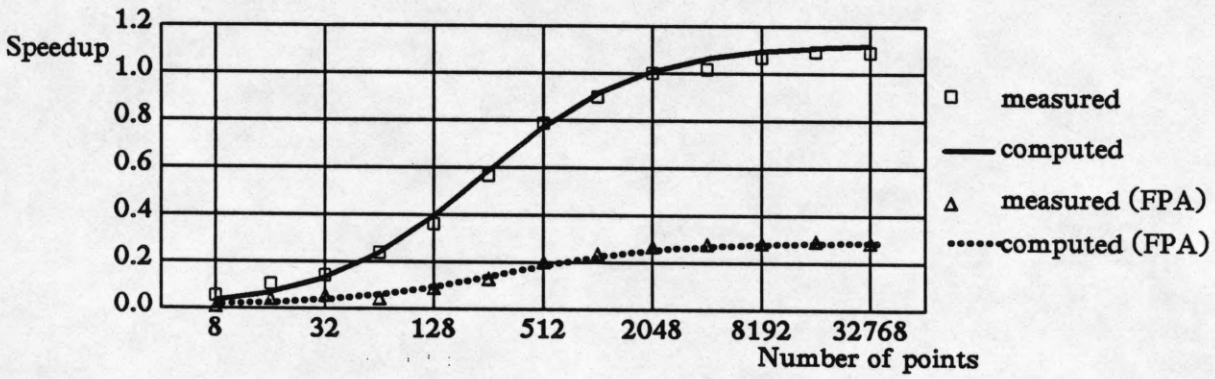

Fig. 3.5. Speedup versus number of points for FFT.

(2) Perform the FFT on each column of the resulting matrix.

(3) Transpose the result of the second step.

In the distributed algorithm, we simply partition the rows (same for columns) between the participating processors and perform the FFT on the rows in parallel.

The size of the problem is $n = u^2$ and the single processor cost of the FFT of the rows is (FFT of u rows, u elements each):

$$t = u(\alpha_0 u \log u) = \alpha n \log n \tag{3.17}$$

where $\alpha = \alpha_0/2$. Each secondary processor receives and sends u/P rows:

$$S = R = \frac{u^2}{P} = \frac{n}{P}. \tag{3.18}$$

The computation cost function on the secondary processors is (FFT of u/P rows, u elements each):

$$\tau = \frac{\alpha}{P} n \log n, \tag{3.19}$$

and the network time function is:

$$t_p = \frac{\alpha}{P} n \log n + 2(P-1)C(\frac{n}{P}), \tag{3.20}$$

yielding a speedup of:

$$\sigma_0 = \left(\frac{1}{P} + \frac{(P-1)}{\alpha n \log n} 2C(\frac{n}{P}) \right)^{-1}. \tag{3.21}$$

The time complexity of the 2d-FFT algorithm is:

$$T = \beta n + 2\alpha n \log n, \tag{3.22}$$

where βn accounts for matrix transposition and $2\alpha n \log n$ represents the serial counterpart of the two parallel phases. The serial fraction is:

$$\gamma = \frac{\beta}{\beta + 2\alpha \log n} \tag{3.23}$$

Figure 3.6 shows the measured and predicted speedup of the 2d-FFT problem.

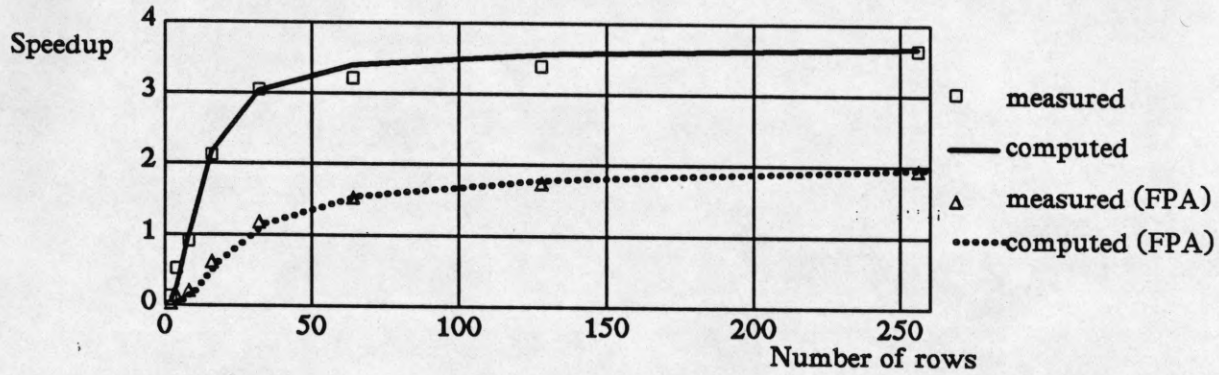

Fig. 3.6. Speedup versus number of rows for 2d-FFT.

3.4. Discussion

The overall speedup, σ, depends on the speedup of the parallel phase, σ_0, and on the serial fraction, γ. For FFT, the speedup of the parallel phase is:

$$\sigma_0 = \left(\frac{1}{P} + \frac{P-1}{\alpha n}[C(1.5n/P) + C(n/P)]\right)^{-1} = \left(\frac{1}{P} + (P-1)\left(\frac{2.5a}{\alpha P} + \frac{2l}{\alpha n}\right)\right)^{-1},$$

which is limited by the communication overhead term, $2.5a(P-1)/(\alpha P)$, regardless of the problem size n. The communication overhead, coupled with a constant serial fraction, $\gamma=(\beta-\alpha)/\beta$, results in a mediocre speedup for FFT even for very large problem sizes.

In the case of 2d-FFT, the parallel phase speedup,

$$\sigma_0 = \left(\frac{1}{P} + \frac{2(P-1)C(n/P)}{\alpha n \log n}\right)^{-1} = \left(\frac{1}{P} + 2(P-1)\left(\frac{\alpha}{P\log n} + \frac{l}{\alpha n \log n}\right)\right)^{-1},$$

contains communication overhead terms which approaches zero for large n. In addition the serial fraction, $\gamma = \beta/(\beta + 2\alpha\log n)$, vanishes for large n. Therefore, the asymptotic speedup is P; however, the speedup approaches its asymptotic value at a slow rate because the function $\log n$ grows very slowly as n increases.

4. Projections

In this section we determine the maximum speedup achievable for the previously described class of algorithms. We then study the effect on performance of 1) adding a communication processor to each workstation and 2) broadcasting data to the secondary processors.

4.1. Maximum Speedup

Given a fixed size problem and a large number of processors, we would like to determine whether or not the speedup peaks and starts declining. If so, we want to determine the number of processors at which this phenomenon occurs. By differentiating the speedup function with respect to P and setting the resulting expression to zero we get:

$$\frac{d}{dP}\tau_0 + (P-1)\frac{d}{dP}(C(S)+C(R)) + \frac{d}{dP}M + C(S) + C(R) = 0. \tag{4.1}$$

Solving the above equation in the range of processors applicable to the given problem, we obtain the number of processors for which the speedup peaks. For example, when applied to matrix multiplication Equation 4.1 yields†:

$$P_{peak} = \left\lfloor \frac{\alpha n^{1.5} - 2an}{an + 2l} \right\rfloor^{1/2}, \tag{4.2}$$

which when evaluated at n=2500 (50 by 50 matrix) results in $P_{peak} = 10$. Figures 4.1 and and 4.2 show the speedup versus the number of processors for a range of problem sizes for matrix multiplication, FFT, and 2d-FFT (the FPA results of FFT is not shown because no speedup is acheived in that case).

† Recall that the single processor computation cost is $\alpha n^{1.5}$ and that the communication cost function is $an + l$ where n is expressed in bytes.

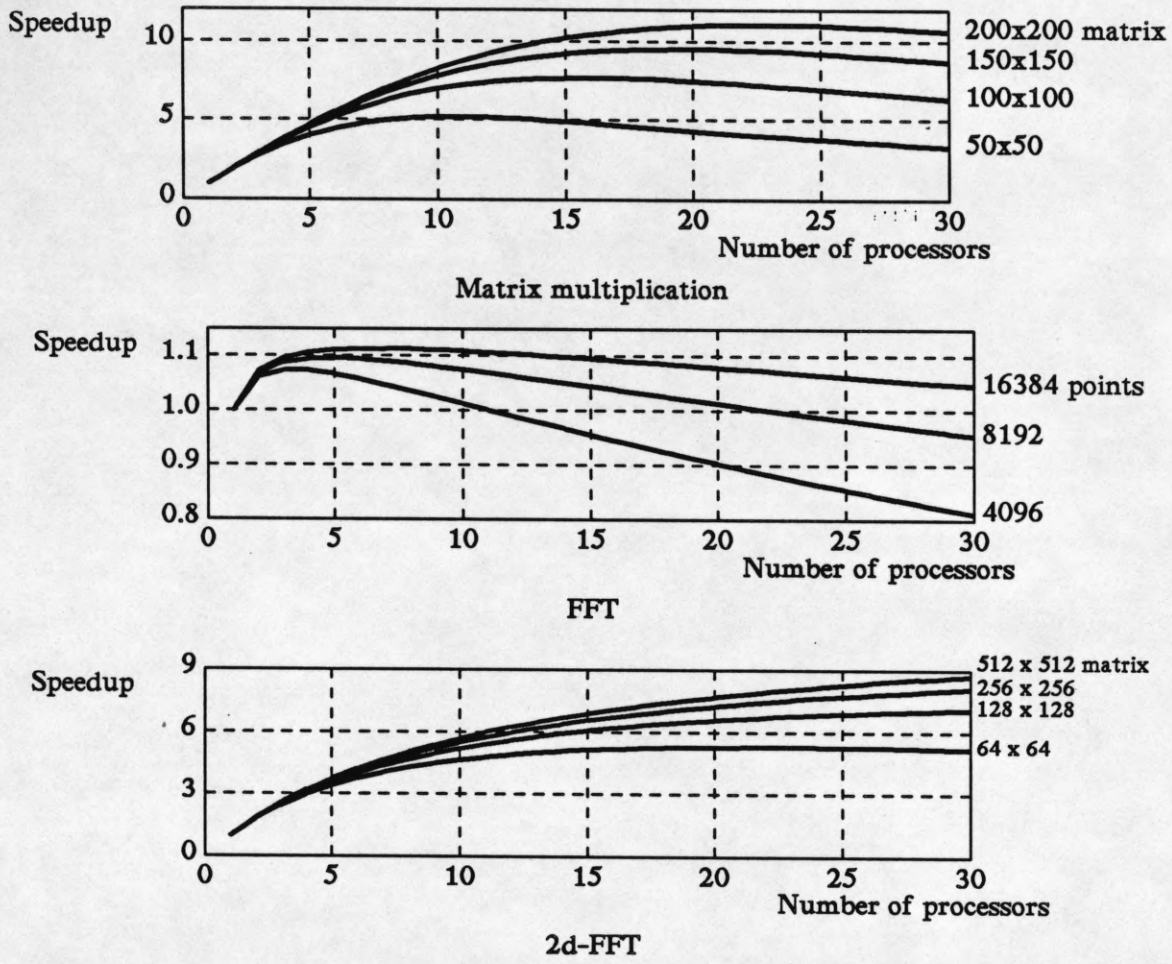

Fig. 4.1. Speedup versus number of processors (no FPA).

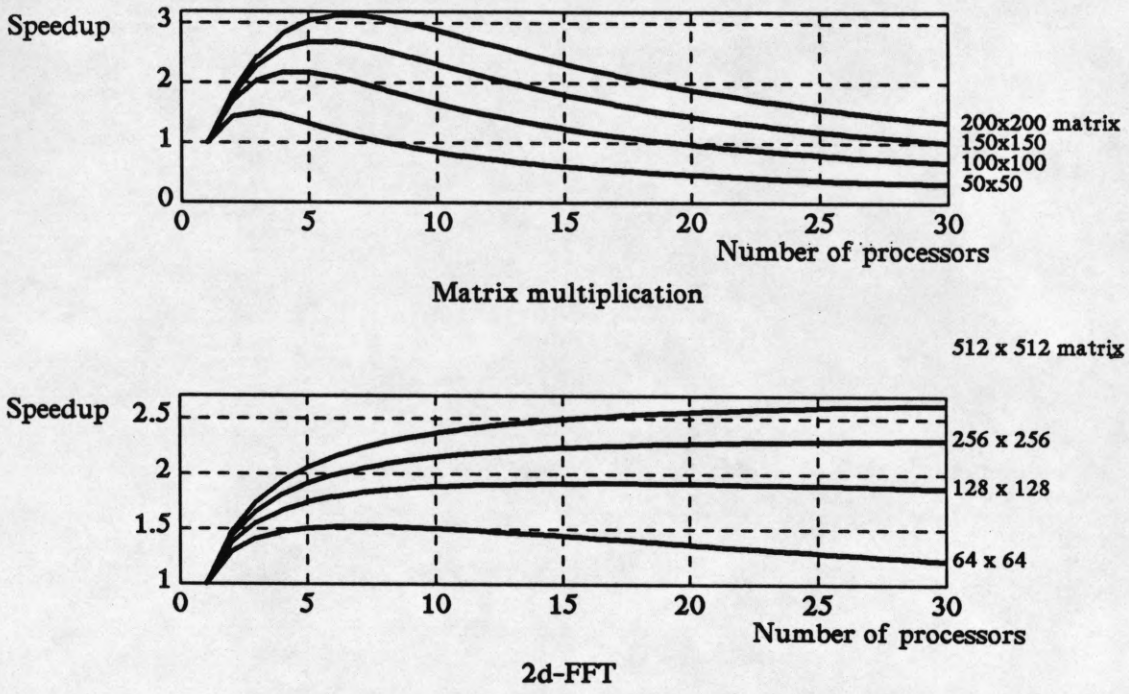

Fig. 4.2. Speedup versus number of processors (FPA).

4.2. Communication Processor

By equipping each workstation with a dedicated communication processor, we can overlap communication and computation and reduce the delays in the system. The operating system would include non-blocking interfaces to the communication processor allowing the computation processor to proceeds with its computation right after transferring data to the communication processor. We recognize two different types of synchronization between the communication and computation processors:

Conservative

The computation processor blocks while data is being transferred to a communication processor buffer. This guarantees that the computation processor cannot corrupt data while it is being transferred.

Risky

The computation proceeds right after passing to the communication processor the address and the size of the data. The communication processor transfers the data directly to the network. This avoids copying the data and blocking the computation processor; however, it suffers the risk of having the computation processor modify the data before the computation processor sends it.

We will assume risky synchronization since we are interested in the maximum achievable speedup. The resulting timing diagram is shown in Figure 4.3 where "processor" refers to a computation processor and "cp" refers to a communication processors. The corresponding speedup equation becomes:

$$\sigma = \left(\frac{\tau}{t} + \frac{(P-1)C(S) + C(R)}{t} \right)^{-1}. \tag{4.3}$$

The communication processor reduces the communication overhead factor $(P-1)C(R)$ to $C(R)$; however, this reduction need not translate into a significant speedup boost since the reduced overhead is only a portion of the overall overhead $(P-1)(C(R)+C(S))$. Figures 4.4 and 4.5 shows the resulting effect on speedup. Matrix multiplication shows no significant improvement in speedup since the size of data received by the primary processor, $R=n/P$, is only a small fraction of that sent to each secondary processor, $S=n+n/P$. In the case of FFT, a more visible increase in speedup

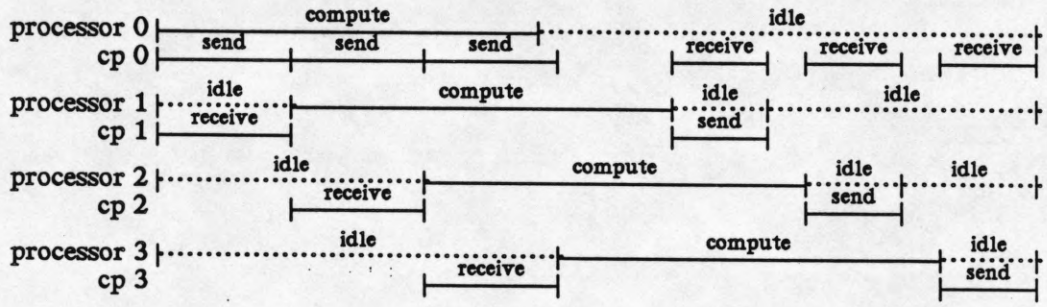

Fig. 4.3. Timing diagram in the presence of communication processors.

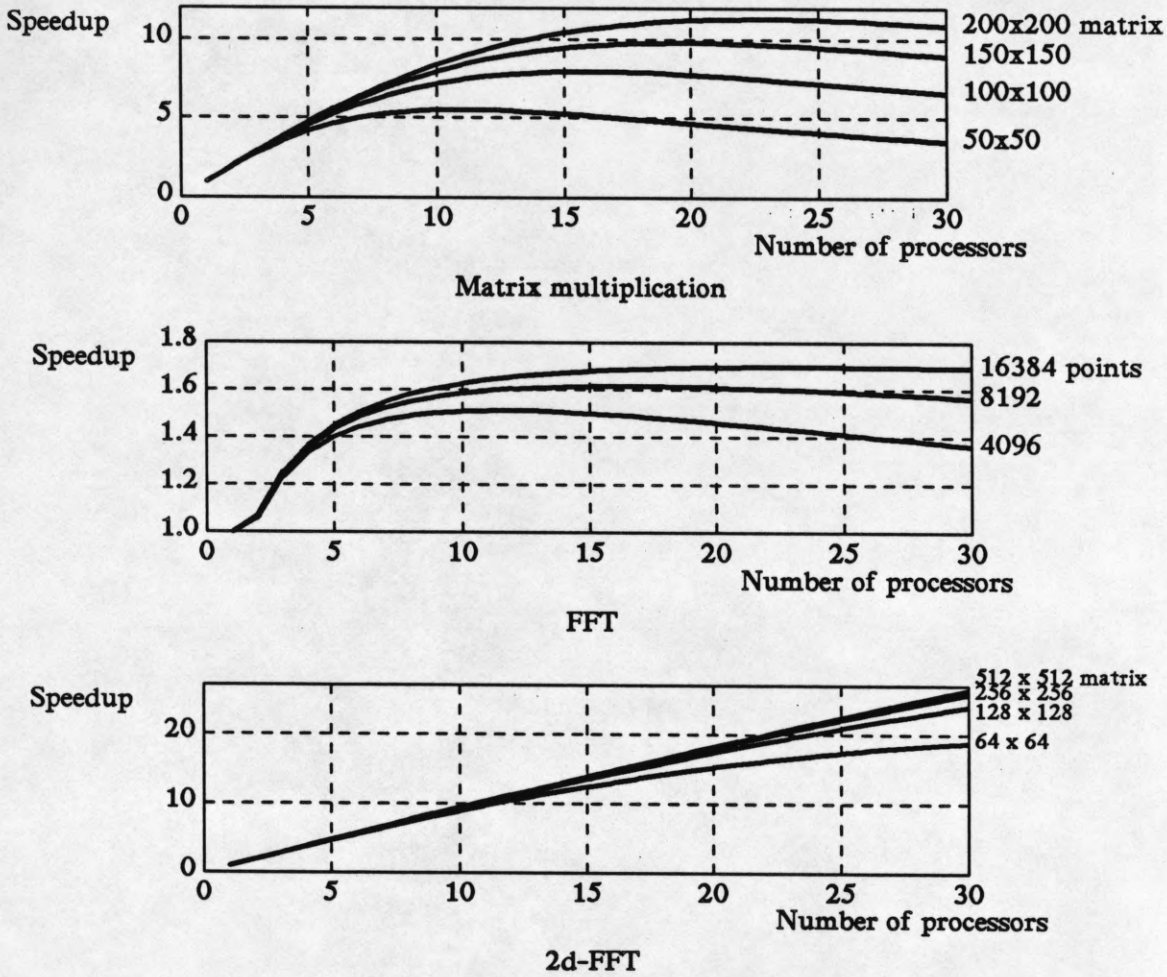

Fig. 4.4. Speedup vs. number of processors using communication processors (no FPA).

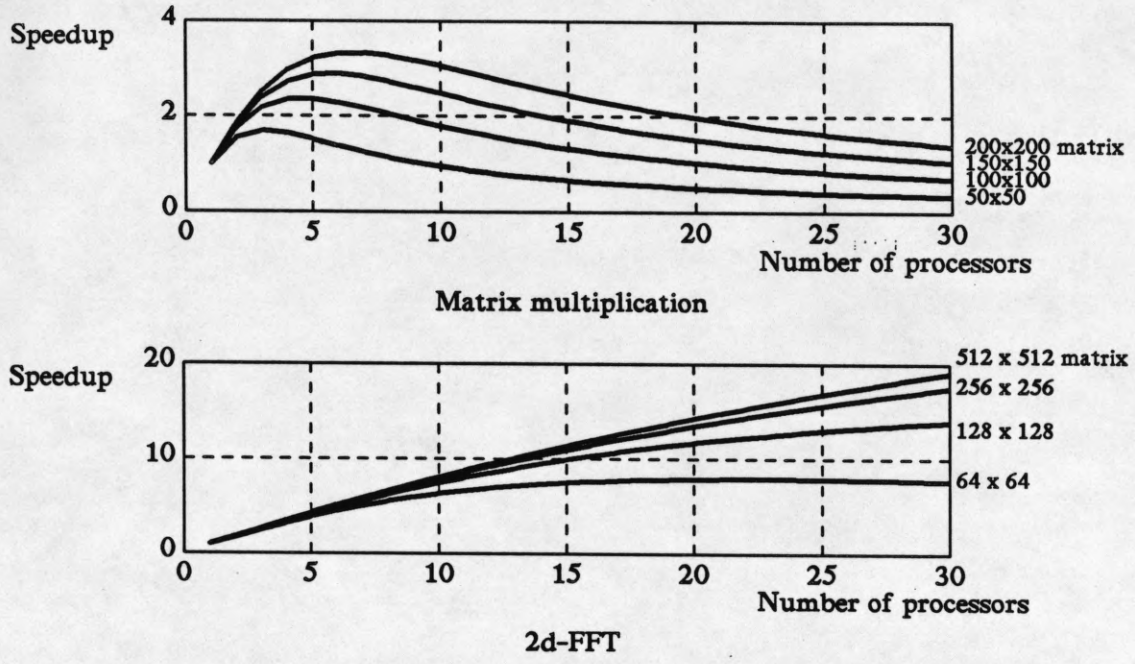

Fig. 4.5. Speedup vs. number of processors using communication processors (FPA).

results because the size of data received, $R=n/P$, is a significant fraction of the size of data sent, $S=1.5n/P$. For 2d-FFT the size of data sent, n/P, is identical to that received. Reducing $(P-1)C(R)$ to $C(R)$ almost halves the overall communication overhead dramatically improving the speedup of 2d-FFT.

4.3. Broadcasting

Broadcasting any common data to all the secondary processors reduces the communication time by reducing the size of the transferred data. In addition, broadcasting all data (common or otherwise) to all secondary processors and having each secondary processor disregard the portions of the data it does not need, incurs the communication latency overhead once instead of $P-1$ times. Figure 4.6 shows the timing diagram for broadcasting. The corresponding speedup equation is:

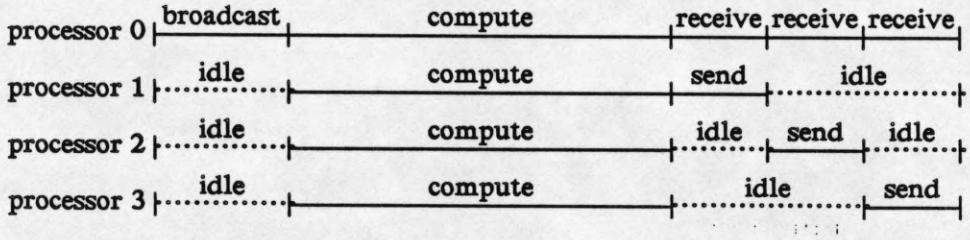

Fig. 4.6. Timing diagram for broadcasting.

$$\sigma = \left[\frac{\tau}{t} + \frac{C(B) + (P-1)C(R)}{t}\right]^{-1},$$

where B is the size of the broadcast data and R is the size of the data received from each secondary processor. With broadcasting, the overhead for sending the data changes from $(P-1)C(S)$ to $C(B)$. In the worst case, no common data is sent to the secondary processors, and the size of the broadcast data is $B=(P-1)S$; the only saving in this case is in the communication latency which is incurred once instead of $P-1$ times. However, the saving in communication latency is outweighed by the delay incurred by each secondary processor in the reception of all of the broadcast data. Both FFT and 2d-FFT fit the worst case scenario and, consequently, show no improvement in speedup from broadcasting. In the best case, the same data is sent to all the processors and the size of the broadcast data, B, is identical to that sent to each secondary processor in the serial communication case, S. Thus the cost of sending the data is reduced by a factor of $P-1$. Matrix multiplication comes close to the best case since a sizable portion of the data sent to the secondary processors is common. Figures 4.7 and and 4.8 show the speedup versus the number of processors when broadcasting is used without and with the FPA, respectively.

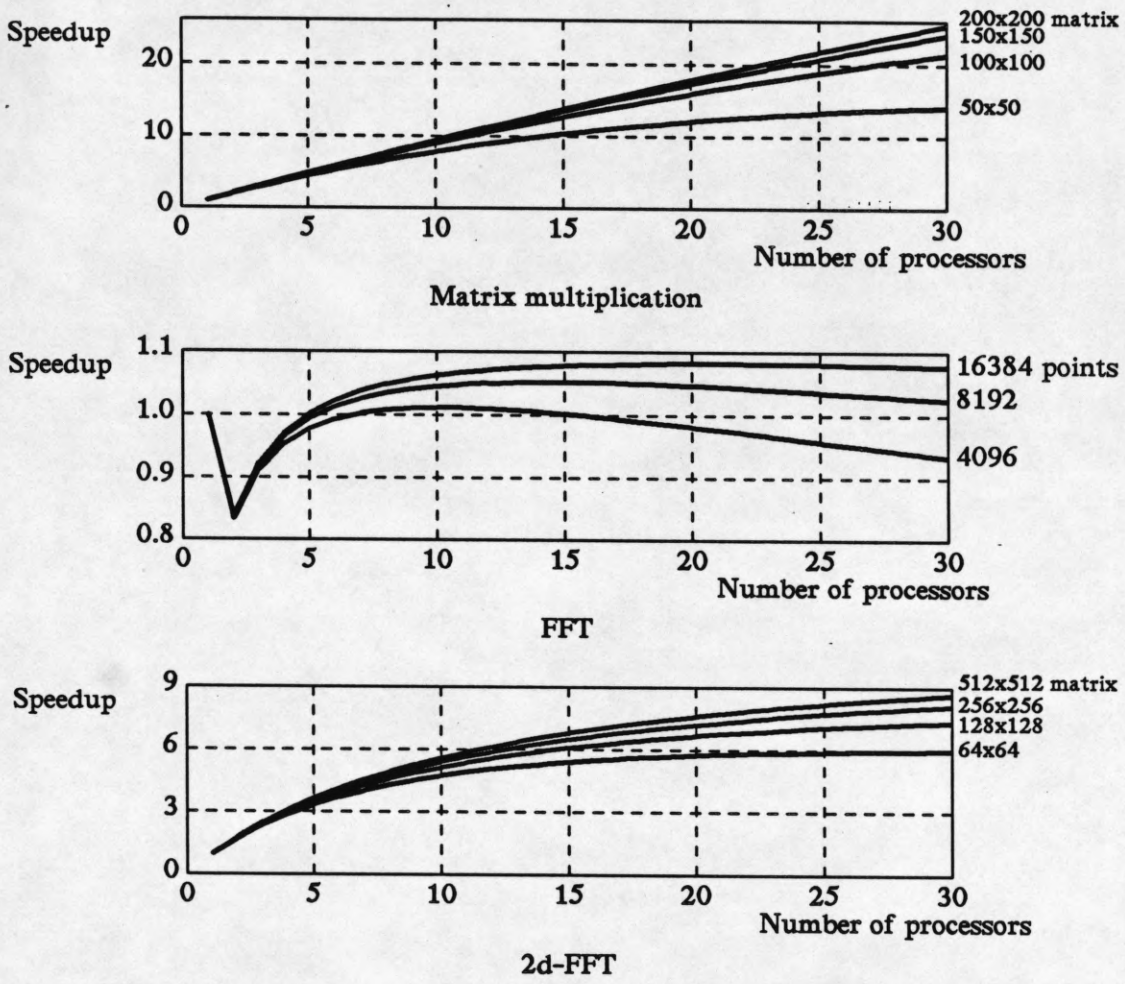

Fig. 4.7. Speedup versus number of processors for broadcasting (no FPA).

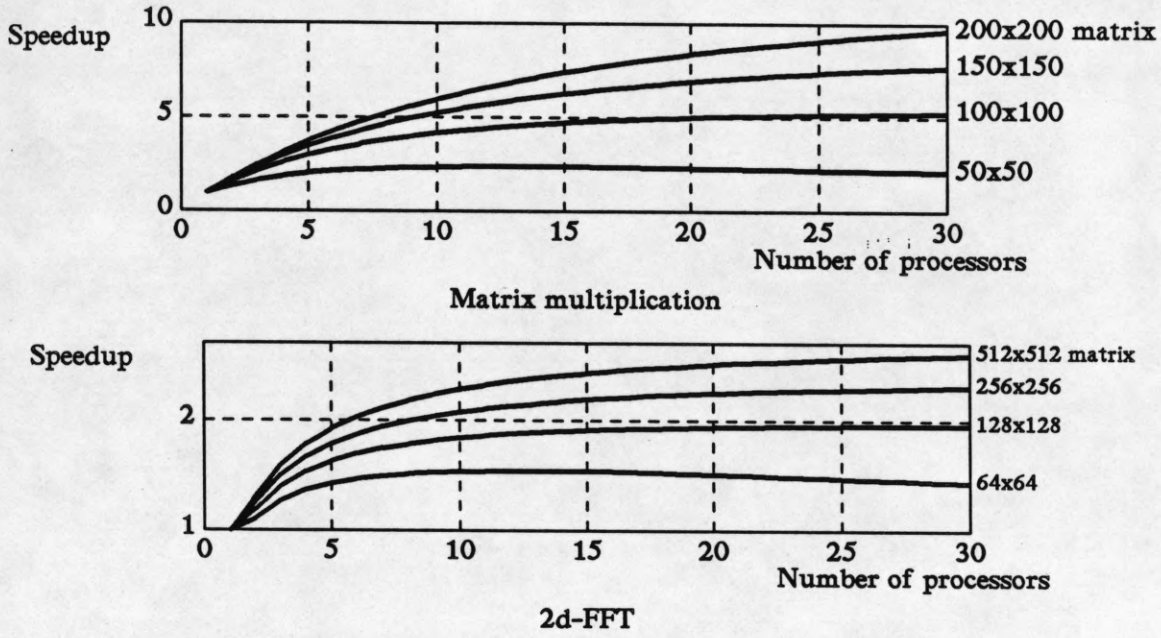

Fig. 4.7. Speedup versus number of processors for broadcasting (FPA).

5. Conclusions

In this report we presented a performance model for a loosely coupled cluster of workstations and for a class of parallel algorithms. We validated the model by comparing its results to empirical data. We used the model to study the effects of varying the system parameters. We showed that the speedup is limited by the large communication overhead. In addition, the large communication overhead may result in a decrease in speedup as more processors are directed to a given problem. We also showed that broadcasting substantially improve the speedup in the case where a large portion of the broadcast data is used by each receiving processor. The use of a communication processor is beneficial in the case where most of the data received by each processor is different that that received by any other processor.

REFERENCES

[1] J. Stankovic, "A Perspective on Distributed Computer Systems," *IEEE Transactions on Computers*, vol. C-33, pp. 1102-1115, December 1984.

[2] P. Heidelberger and S. Lavenberg, "Computer Performance Evaluation Methodology," *IEEE Transactions on Computers*, vol. C-33, pp. 1195-1230, December 1984.

[3] B. Lint and T. Agerwala, "Communication Issues in the Design and Analysis of Parallel Algorithms," in *IEEE Transactions on Software Engineering*. pp. 174-188, March 1981.

[4] D. Vrsalovic, E. Gehringer, Z. Segall, and D. Siewiorek, "The Influence of Parallel Decomposition Strategies on the Performance of Multiprocessor Systems," in *International Symposium on Computer Architecture*, Boston, pp. 396-405, June 1985.

[5] D. Gannon and J. Rosendale, "On the Impact of Communication Complexity on the Design of Parallel Numerical Algorithm," *IEEE Transactions on Computers*, vol. C-33, pp. 1180-1194, December 1984.

[6] M. Dubois and F. Briggs, "Performance of Synchronized Iterative Processes in Multiprocessor Systems," *IEEE Transactions on Software Engineering*, vol. SE-8, pp. 419-431, Juy 1982.

[7] W. Stallings, *Data and Computer Communications*. New York, New York: Macmillan Publishing Company, 1985.

[8] A. Oppenheim and R. Schafer, *Digital Signal Processing*. Englewood Cliffs, New Jersey: Prentice-Hall, 1975.

[9] L. Jamieson, P. Mueller, and H. Siegel, "FFT Algorithms for SIMD Parallel Processing Systems," *Journal of Parallel and Distributed Computing*, vol. 3, pp. 48-71, March 1986.